2016-03

Detection and classification of Web robots with honeypots

McKenna, Sean F.

Monterey, California: Naval Postgraduate School

NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

DETECTION AND CLASSIFICATION OF WEB ROBOTS WITH HONEYPOTS

by

Sean F. McKenna

March 2016

Thesis Advisor: Neil Rowe
Second Reader: Justin P. Rohrer

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			*Form Approved OMB No. 0704–0188*
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington, DC 20503.			
1. AGENCY USE ONLY ***(Leave blank)***	**2. REPORT DATE** March 2016	**3. REPORT TYPE AND DATES COVERED** Master's thesis	
4. TITLE AND SUBTITLE DETECTION AND CLASSIFICATION OF WEB ROBOTS WITH HONEYPOTS			**5. FUNDING NUMBERS**
6. AUTHOR(S) Sean F. McKenna			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A			**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. IRB Protocol number ____N/A____.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited			**12b. DISTRIBUTION CODE**

13. ABSTRACT (maximum 200 words)

Web robots are automated programs that systematically browse the Web, collecting information. Although Web robots are valuable tools for indexing content on the Web, they can also be malicious through phishing, spamming, or performing targeted attacks. In this thesis, we study an approach to Web-robot detection that uses honeypots in the form of hidden resources on Web pages. Our detection model is based upon the observation that malicious Web robots do not consider a resource's visibility when gathering information. We performed a test on an academic website and analyzed the honeypots' performance using Web logs from the site's server. Our results did detect Web robots, but did not adequately detect the more sophisticated robots, such as those using deep-crawling algorithms with query generation.

14. SUBJECT TERMS World Wide Web, crawlers, robots, malicious, logs, honeypots			**15. NUMBER OF PAGES** 63
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UU

NSN 7540–01-280-5500 Standard Form 298 (Rev. 2–89)
Prescribed by ANSI Std. 239–18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

DETECTION AND CLASSIFICATION OF WEB ROBOTS WITH HONEYPOTS

Sean F. McKenna
Civilian, Scholarship for Service
B.A., The Colorado College, 1996

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
March 2016

Approved by: Neil Rowe, Ph.D.
Thesis Advisor

Justin P. Rohrer, Ph.D.
Second Reader

Peter J. Denning, Ph.D.
Chair, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Web robots are automated programs that systematically browse the Web, collecting information. Although Web robots are valuable tools for indexing content on the Web, they can also be malicious through phishing, spamming, or performing targeted attacks. In this thesis, we study an approach to Web-robot detection that uses honeypots in the form of hidden resources on Web pages. Our detection model is based upon the observation that malicious Web robots do not consider a resource's visibility when gathering information. We performed a test on an academic website and analyzed the honeypots' performance using Web logs from the site's server. Our results did detect Web robots, but did not adequately detect the more sophisticated robots, such as those using deep-crawling algorithms with query generation.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

I would like to thank my wife, Kimberly, as well as Dr. Rowe for their Zen-like patience with me.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A Web robot is as an automated program that systematically browses the Web to collect information. Web robots were first developed as tools for search engines to index Web pages, RSS feeds, and other kinds of content on the Internet. Since their inception, the number and diversity of these programs crawling the Web has grown dramatically. Ten years ago the vast majority of traffic seen by Web servers was being generated by human beings [1]. Today, more than half of all traffic seen by Web servers can be attributed to Web robots, which are sometimes called crawlers [2].

The rapid rise in these kinds of programs has been attributed to the explosion in content and user-generated social media on the Internet. The Web search engines like Google require large numbers of automated bots on the Web to build their indexes. Furthermore, the growth of internet has produced a market for businesses, both legitimate and otherwise, which are able to harvest data from the Web quickly and effectively. To keep up with this increasing demand of data mining on the Internet, modern Web crawlers have evolved into sophisticated, multifunctional programs that use complex algorithms for filtering and collecting data [3].

Despite their large and growing numbers, Web robots still roam the Internet in an unregulated manner. While many provide valuable services for Internet users and content providers, a significant portion of bots are intended for unlawful purposes. A recent study [2] found that close to half of all bots are “bad,” or have been reported as abusive. In addition, studies have shown that traffic generated by good or bad bots, even at normal levels, can negatively impact a Web server’s performance [4].

Yet even given the known problems associated with Web robots, Web-server administrators not generally not inclined to ban all bot traffic from their websites. It is difficult to distinguish between good bot traffic, malicious bot traffic, and the human traffic visiting a site (this aspect of bot detection is discussed in detail in Chapter II). Yet the “good” bots are the primary means by which a website’s content gets provided to search engines and other legitimate data collection sources. An outright ban of these

programs prevents the website from providing its content to search engines. Additionally, an outright ban on crawling activity can potentially lead to a server getting blacklisted by legitimate search engines. Legitimate search companies sometimes often associate Web server with crawling bans to websites hosting content that is illegitimate, illegal, or containing malware. For this reason and the others mentioned, there has been a push in research to develop tools that can detect and distinguish the bad from the benign Web robots.

The bulk of the early research on Web traffic focused on the patterns and properties of human traffic [5]. Research in human traffic patterns helped us further understand how people navigate the Web, but did little to explain the behavior of Web robots. Web robots navigate the Web in fundamentally different ways. Humans retrieve information from the Web through standard interfaces like a Web browser. Web browsers, in turn, produce predictable sequences of resource requests, which are based upon a website's design. By contrast, Web crawlers traditionally do not use a standard interface for resource requests to a Web server. These decisions are pre-programmed into and generally do not take into account the website's design or the "human" interface displayed in the browser.

Most recent studies have attempted to understand Web robot traffic through resource request patterns in access logs [6]. There are limitations to this method of traffic analysis. Modern bots possess a range of functionality and navigational strategies that are hard to categorize with log analysis alone. Existing research in Web crawler traffic has measured individual crawler variations in navigation techniques and resources used. Such studies for the most part have only provided a generalized picture of the differences between bot traffic and human traffic.

The objective of the thesis is to evaluate the behaviors of Web robots crawling our university's websites. The experiment used a combination of honeypot traps and statistical analysis on the University's Web server access logs. Prior research [7] has demonstrated that by placing hidden resources within a Web Interface, a primitive honeypot can be created which reliably detects resource requests for individual crawler sessions. We developed and tested our own version of a Web crawler honeypot using this

technique. For the first part of the study, we developed a test site to create a training set of data to demonstrate the feasibility of our technique. We ran several well-known Web crawling scripts and programs on a test website. It was a duplicate of the Naval Postgraduate School Library website where the actual honeypot was placed for the second part of the study. This training set of data from the controlled test informed the placement location and types of resources used for the second test on the Naval Postgraduate School Library website.

The thesis has been organized as follows. Chapter II is an introduction to Web robots and a look at the current landscape of several kinds of bots that crawl the Web today and the security issues they present. Additionally, Chapter II presents a high-level overview of common Web crawling programs. Chapter III reviews prior research on Web robot behavior and detection. Chapter IV introduces our methods for detecting Web crawlers and outlines the process we used for the training data set and live tests on the Naval Postgraduate School (NPS) Dudley Knox Library website. Chapter V presents a detailed evaluation of the findings. Chapter VI summarizes the contributions of this thesis and proposes new avenues for future research.

THIS PAGE INTENTIONALLY LEFT BLANK

II. WEB ROBOTS

The use of scripts to automate the collection of data over the Internet is nearly as old as the Internet itself. Over the years there have been many different terms used to describe these automated programs. The earliest of these programs were known as spiders, and were developed solely for the purpose of indexing websites for search pages. Now spiders are typically called Web crawlers, and have evolved to include such programs such as Web scrapers for harvesting content, link checkers for verifying websites, or site analyzers for archiving Internet data.

A Web robot, or simply a bot, is the term used to describe any type of program which automatically and recursively traverses websites. The most common kinds of Web robots on the Internet are the Web crawlers. Another type Web robot is a scanner, which is similar to a crawler, but designed to search specifically for a website's vulnerabilities.

The "good" Web robots that crawl the Web are an integral part of the Internet, and have played an important role in its evolution and growth. Conversely, the "bad" Web robots have been and continue to be a significant problem. Bad Web robots are widely used for launching denial of service (DDoS) attacks, to automating spam campaigns, for corporate espionage, and performing vulnerability scans of websites on a large scale.

The landscape of the variety and number of Web robots on the Internet has changed in recent years. This chapter looks at the existing landscape of bot traffic on the Web and well as some contemporary examples of malicious Web robots.

A. RISE OF THE BOTS

This growing proportion of traffic that comes from Web robots presents a problem for the performance of Web servers. Researchers in 2013 [4] examined the Web logs of the university's Web servers over a two-year period. The findings suggested that the resource requests of Web robot traffic were more likely to create cache problems on Web servers than those coming from human traffic. Web robots were more likely to request large resources than human beings did, and did not use local caches in Web

browsers. Web robot traffic also followed a different resource-popularity distribution than human traffic and thus negated the predictive cache algorithms of Web servers that were tuned to human requests. These findings were supported by Koehl et al. [8], who found that Web crawlers in their study represented "6.68% of all requests, but consumed 31.76% of overall server processing time." These studies suggest a need for more robust techniques and tools to identify and block unwanted Web robots from Web servers. Furthermore, if trends continue, the problem of "crawler overload" will continue hinder performance of Web servers.

B. BAD ROBOTS

The ability to detect and differentiate different Web robots is critical if we are indeed serious about protecting our application layer. There is limited research available on how bots are used maliciously and the ways in which administrators and legal entities can respond to them. Knowing the history and the diversity of Web robots is important if we are to protect Web servers against bot-related threats.

1. Distributed Denial of Service Attack (DDoS)

For years distributed denial of service attacks (DDoS) have been one of the biggest threats to network security. A DDoS attack is a coordinated attack on the availability of some type of network, usually occurring with the unknowing aid of a large set of compromised computers. In recent years, there has been an increasing threat in the kind of DDoS attacks that occur in the application layer or layer 7 using Web robots. According to a 2015 "State of the Internet Security" report by Akamai [9], the number of DDoS attacks had doubled in 2015 from the number of attacks in 2014. The percentage of these attacks that were conducted from the application layer was 9.32%.

A 2014 study by Qin et al. [10] analyzed the attack behaviors of layer 7 DDoS attacks on websites and classified them into four different categories. They were:

- The single URL—an attack that sends repeated requests to the target at fixed or random speeds.
- The multiple URL—a repeated attack that sends a request to a fixed group of URLs in the attack vector.

- The random attack—sends requests to URLs randomly from the attack space of an already scanned website.
- The session attack—those attacks in which the bots have been trained on the victim website and attempt to mimic a human's normal browsing behavior.

The same study tested whether using a model of human resource request traffic could be used to detect these four types of DDoS attack in simulation. The results were encouraging. However, the authors concluded that these post-mortem log analysis techniques would be ineffective in a real-word environment. Using post-mortem log analysis to preventing DDos attacks took too much time. It was proposed that future research investigate the reduction of "the span of time window" and to "consider real-time detection" [10]. Real-time detection is a reoccurring theme for this area of security research. The ability of a Web server to identify and interrupt the active sessions by malicious Web robots is paramount to prevent a server-side exploit or DDoS attack from causing damage.

2. Web Scanners

Web scanners are another common form of Web robot. Scanners crawl selected websites or applications to look for known security vulnerabilities. A large number of commercial and open-source Web scanning tools are available on the market. Web scanners are used by white-hat (defensive) security engineers to test for common Web-application vulnerabilities such as SQL injection, cross-site scripting, command execution, and directory traversal. Scanners are also employed by bad actors looking for opportunities for later intrusion attempts or exploitations. It can be difficult to detect Web scanners as they are often stealthy and usually only scan a few portions of a website at a time [11].

The traffic footprints of scanners are different from the traffic footprints of a typical search bot. The number of requests sent by Web scanners is small. Scanners tend to avoid requesting common files such as images or documents that do not pertain to the security profile of the target. Additionally, scanners tend to have more 404 request (page

not found) errors than traditional bots since they will probe for hidden resources of their target by guessing resource locations.

In 2013, a website for the Department of Homeland Security was broken into by a hacktivist group using Web scanners that uncovered a vulnerability on the site. The hackers discovered that the website *studyinthestates.dhs.gov* contained a directory-transversal vulnerability that allowed them access files that should have been inaccessible. The directory-transversal issue was a known problem with the site's content management system, WordPress, and the version of PHP being used on the site's Web server. A timely patching of the site's content management software and its Web server could have easily prevented this [12].

In a 2014 study, Xie at al. [11] found that the use of Web scanners were widespread use on university websites. The researchers looked at university network traffic at the University of California Riverside and discovered that they were receiving an average of 4000 scans from unique IPs per week. The scanner traffic favored *php*, *asp*, and *zip* files along with common login and administrator filenames for well-known Web applications. As with the scans in the DHS breach, a significant portion of the scans on their websites were using probes which looked for common content management system vulnerabilities.

3. Web Scrapers

Web scrapers are similar to the Web crawlers except that scrapers are designed to mine the unstructured data from the Web in a way that can be stored and evaluated at a central database. This is accomplished through querying a Web server, requesting data in the form of Web pages and other resources, and parsing the data to extract needed information. Web scrapers employ a wide variety of programming techniques and technologies, including data analysis [13].

Web scraping has been a popular technique for startups and Internet researchers to gather data from the Internet. It is an inexpensive and powerful way to gather data from websites without the owner's partnership or consent. Web scraping was largely left out of the courts until 2010 when the company eBay filed an injunction against the company

Bidders Edge, which had been scraping the eBay website to provide information to their own customers bidding on eBay's website. The U.S. courts acknowledged that companies using "scrapers" or "robots" could be held liable for committing trespass to chattels [14]. Although the eBay case set a legal precedent in the United States, the practice of website scraping has remained largely unabated. In 2012, the startup called 3Taps used Web scrapers to pull ads from the website Craigslist. Craigslist later successfully sued on the grounds that 3Taps had violated the Computer Fraud and Abuse Act [15].

Web scraping made the headlines more recently in 2015 when the financial news company Selerity posted Twitter's earnings report (which missed the target) before the close of trading day and before it had been released by Twitter. Selerity's Web scanner had discovered the link to the Twitter quarterly earnings announcement ahead of its official release by using a Web scanning algorithm to guess potential query strings for the news release. This publicized incident did result in a revision of the company's "security by obscurity" method of handling press releases on the website [16].

Web scraping can also be used maliciously by harvesting content from a victim's site then publishing it under a new site with a new name. The demand for companies to protect their online assets from Web scrapers has produced several commercial anti-scraping services (e.g., Distil Networks, ScrapeShield, BotDefender). These commercial services typically do not provide the technical details as to how their products defend against unwanted scraping.

4. Web Crawlers

Web crawlers are the automated programs used by search engines to explore the Internet and download data. While many of these programs are used by legitimate search engines, crawlers can be used to gather data for malicious purposes. Furthermore, legitimate crawlers have also been hijacked to be used for illegitimate purposes. In 2013, Daniel Cid discovered that his Web application was blocking requests from Google-owned IP addresses [17]. He determined that the IP was being blocked because the address had appeared, from the firewall and other logs, to be attempting a SQL injection. A SQL injection is a commonly used technique for exploiting insecure Web applications.

Google Web crawlers were attempting SQL injections on his website using Google crawlers as a proxy. The hackers created Web pages with the SQL injections contained in the URL strings embedded on their Web pages. The Google crawlers followed the URLs on the hackers' website to the victim site's that was included with the SQL injection in the URL. This simple attack demonstrated how search engines can be susceptible to their "good" crawlers getting hijacked by bad actors [18].

III. WEB ROBOT DETECTION

The vast majority of the research in robot detection has relied on crawling artifacts such as fake user-agents, suspicious referrers, and bot-like traffic patterns. Researchers have looked into a variety of log analysis techniques for bot detection, including the use of machine learning algorithms to identify Web robots [19]. Stassopoulo et al. were the first to introduce probabilistic modeling methods for detecting Web robots using access logs. These researchers evaluated behaviors such as click rate, number of image requests, http error responses, and robots.txt requests to build their model [20].

There is a significant drawback to detection methods that passively model Web traffic: they are unable detect Web robot sessions as they occur. However, there exists some detection techniques that operate in real time and do not depend on offline logs analysis. These techniques provide a test to the user while the session on the website is active, and analyzes the response to determine whether a human or a robot has produced it. Such programs are called Turing tests.

A. TURING TESTS

Research in distinguishing computers from humans started with Turing tests [20]. The early Turing tests asked if a human could distinguish between a human or machine by asking questions to both. In contrast, modern Turing tests use computer programs to perform a test that distinguishes between another computer program and a human.

Turing tests are designed to be performed easily by humans and to be a challenge for computers. To detect Web robots, a Turing Test is given to a website's visitor to determine whether the session has been generated by a robot or human. The most well-known of these online tests are CAPTCHAs (Completely Automated Public Turing Tests to Tell Computer and Human Apart). CAPTCHA tests use a form of visual letter recognition to determine if a visitor is human or robot. Although they have long track record of success, in recent years the effectiveness of CAPTCHAs has waned to the growing sophistication of attackers using scanning tools to decode the images [13]. As a

result, CAPTCHA tests are being produced with more distorted images to make it more difficult for Web robots to interpret. Website administrators and designers however, are faced with a dilemma of whether to use the more difficult CAPTCHAs tests. If the tests are too difficult, there is a fear that users will get discouraged and may terminate the session [20].

While sophisticated Web robots can be programmed to exhibit human-like traffic patterns to avoid detection, these bots can still be detected using the automated Turing tests. A kind of Turing test that automatically distinguishes a Web robot from a human is called a Human Observational Proofs. HOPs are able to differentiate bots from humans by identifying bot behaviors that differ intrinsically from human's behaviors. For example, a HOP can determine the whether a session is bot or human by looking at Web browser-based characteristic such as keystrokes and mouse movements.

A 2006 study [21] introduced this technique using involved mouse movement and keyboard clicks to determine if the user was human. The experiment involved embedding a unique JavaScript into every new Web page requested and using an event handler to detect if any mouse or keyboard activity occurred within a session. If any mouse or keyboard activity was recognized the session was recorded as human. If there was no mouse or keyboard activity, it was recorded as a bot. It was found that this simple technique could reliably predict whether a human or robot started the session. To account for the fact that some people browse with JavaScript disabled on their browsers, the researchers embedded a second HOP test that involved hiding a resources in the html, like a honeypot. The results of the study showed that when using these combined techniques, human users could be detected at a 95% rate of accuracy were detected with a 2.4% false positive rate.

While this detection system proved reliable at the time, there are some inherent drawbacks to using this technique for detecting modern Web robots. Modern Web robots are able to emulate a browser interface and executing JavaScript and Cookies on a Web page. There are packages in Python such as PhantomJS capable of executing the JavaScript as the protagonist executes their Web bot code the website. It performs this

actions this by loading the website into memory and executes JavaScript on the page [13].

Despite this known countermeasure, many Web applications today used this HOP technique for preventing bots from doing things such as submitting forms. A Web server using these types of controls will accept form submission only when a user's mouse or keyboard event has been detected in the form submission process. Unfortunately, the authors of bots have successfully adapted to this security control. By creating automated bots which mimic these behaviors [22].

Web robot developers have created an advanced form of bot which can open a Web page in a browser using Operating-system API calls to generate keyboard click and mouse movements. Researchers have studied these bots to determine if their mouse and keyboard patterns were predictable and differed from human mouse and keyboard behavior. They discovered that the bot behavior when simulating mouse and keyboard events were predictable, and could be employed to produce a detection rate of over 99% with limited processing overhead [22].

B. HONEYPOT

A *honeypot* is a security mechanism designed to make it easier to detect someone or something that is using a system without authorization. Honeypots are a popular method for detecting unwanted visitors on websites. There are two kinds of honeypots: the active kind that attempts to "trap" unwanted traffic, and the passive kind, that is used solely for detection. A kind of popular trapping honeypot is a "spider trap" which tricks Web crawlers into infinite loops of page requests. A spider trap can be created in several ways, such as by adding links with infinitely deep directory trees, or by using dynamic pages with unbounded parameters. However, a drawback is that legitimate Web crawlers can be fooled by these traps as well. As a result, Web administrators which employ spider traps will often include their spider trap pages on the exclusion list of a robots.txt file [13].

A popular kind of "passive" honeypot is hiding links and other resources within the Web page using formatting. A human which uses a browser to read a Web page cannot see links or other resources that have been hidden from view by the site's design.

However, a Web crawler only looks at a page's source code and will not likely check the formatting to see if a resource is hidden from view before requesting it. This technique can be applied to any element in a Web page.

Honeypots are a useful tool for Web robot detection and prevention. It is possible for a website when honeypot is visited, to activate a server side script that blocks the visitor's IP address, or logs out the visitor. This technique is currently being employed by commercial anti-scraping tools on the market. Although commercial anti-scrapping tools do not publish technical details on their detection techniques, it appears that these techniques are active component in their detection platforms.

However, these honeypot defense techniques can be thwarted if Web robots are employing certain countermeasures. As discussed earlier, sophisticated Web robots can activate cookies, execute JavaScript code, and even emulate keyboard and mouse movements to avoid detection. In addition, Web robots can employ algorithms that search a Web page's style sheet (CSS) for formats by which a resource can be hidden from view. If a Web robot finds a page element with a hidden field tag, it is able to exclude the resources with that style from its crawl [13].

Currently there is little research that looks at the effectiveness of simple honeypot techniques for detecting both legitimate and illegitimate Web robots. There are legitimate reasons for a Web site designer to hide content within a Web page. Hidden fields can be used in Web forms to store default values which can be dynamically altered depending on the user's actions. However, in the past, Web developers hid content in Web pages primarily to attract legitimate search engine crawlers to their site. The practice of hiding content within Web pages is used by Web developers to increase a Web page's search engine optimization (SEO), or a higher page rank in search engines. Web page designers pack hidden content viewable only to Web crawlers in Web pages to inflate the page's rankings in search results (keyword stuffing). Search engines have adapted to this practice by modifying crawling algorithms to detect this kind of keyword stuffing [23]. These hiding techniques involve tricks such as setting the size of a page layer to zero, indenting a page element so far as to appear off the screen, and hiding the keyword text by making it the same color as the background of the Web page.

IV. METHODOLOGY

The objective for this experiment was to determine whether a honeypot placed in Web page can effectively detect and classify Web robot traffic. This chapter discusses the methodology of our experiment.

A. MECHANISMS FOR HIDING CONTENT

Content may be hidden for users with visual impairments. In our experiments, it was important that hidden honeypot resources be inaccessible to programs used by visually impaired users. Our assumption was that techniques for hiding content that were explicitly defined would be more likely to be ignored by screen readers than the ones which were not. In contrast, "constructed" formatting techniques such as hiding content through indentation or clipping are more likely be read by screen readers. We tested this hypothesis using the Apple Voice Over screen reading program. We ran the Apple Voice over program on a test page in CSS format with content hidden using six different CSS formatting methods for "hiding" layers. The results (see Table 1) showed that four of the five constructed CSS methods were accessible by our screen reader. Only two rules, the CSS rule with the height and width of zero, and the explicit CSS rule of *display:none*, were ignored by our screen reading test. As a result, we used the explicit *display:none* as a hiding method for our subsequent experiments.

Table 1. Mechanisms for Hiding Content

CSS Rules	Display Effect	Accessibility Effect
display: none;	Element is removed from the page flow and hidden; the space it occupied is collapsed	Content ignored by screen readers
height: 0; width:0; overflow: hidden;	Element is collapsed and contents are hidden	Content ignored by screen readers
text-indent: -999em;	Contents are shifted off-screen and hidden from view, but indentation might not completely hide content on some screens	Screen readers accessed content, but only inline elements.
position: absolute; left: 999em;	Content is removed screen view to the left-hand edge;	Screen readers accessed content
position:absolute; verflow: hidden; clip: rect(0 0 0 0); height: 1px; width:1px; margin: -1px; padding: 0; border: 0;	Content is clipped and collapsed and content in div does not affect overflow.	Screen readers accessed content
z:index: -1	Content is placed behind elements with greater index number. Hides element by visual obstruction within flow of page.	Screen readers accessed content

B. RESOURCE TYPES

Research looking at Web traffic on college Web servers found that Web robots requested different resources than humans [4]. Bots had a higher preference for document files (*xls, doc, pt, ps, pdf, dvi*), Web-related files (*html, hmt, asp, jsp, php, js, css*), and *noe* (no extension) requests. A Web robot which requests a *noe* is attempting to access a directory or directories which is not directly linked from Web pages. As shown in Figure 1, our experiment used the document and Web resources for our honeypot.

Based upon our discussions with the information assurance staff at NPS, we decided to tailor the honeypot's content for bot's targeting email addresses, or "spam" bots. They were interested in knowing whether websites crawled by Web robots were looking specifically for email addresses, and *.mil* domains.

To test for this, we created files of the three most popular content types (*pdf, doc, html*) to be hidden in Web pages. A third of these files contained dummy military email addresses within the text, another third contained only regular email addresses, and another third contained no email addresses, only text (see Table 2).

Table 2. Test Contents Added to the Honeypot

Name	Type	Directory	Content
a_mail_noclass.html b_mail_clas.html	Web	class noclass	Email - Civilian Emails
a_mail_class.html b_mail_class.html	Web	class noclass	Email – Military Emails
a_help.html b_help.html	Web	class noclass	Text only
112508.pdf	pdf	Class noclass	Email – Military Emails
112508.doc	doc	Class noclass	Email – Military Emails
04151795.pdf	pdf	Class noclass	Email - Civilian Emails
04151795.doc	doc	Class noclass	Email - Civilian Emails
112507.pdf	doc	class noclass	Text only
112507.doc		class noclass	Text only
Clfd.php	php	class	Text only

C. SANDTRAPS

To show that honeypots can be used a part of a larger intrusion-detection or intrusion-prevention framework, we created a script to capture bot resource requests in real time. This script was called a sandtrap. Our initial foray used JavaScript placed into the two HTML honeypot resources to record the time, IP Address, and User Agent string of each requestor of the HTML file. In the initial testing, the scripts did record human requests but not crawlers. Unfortunately, most Web crawlers and Web scrapers in our test did not execute JavaScript when crawling. Therefore, we shifted the efforts towards implementing a server-side PHP script to catch crawlers because the NPS site was developed in PHP. Our PHP code (Appendix A) logged the time, IP Address, and User Agent strings of the visitor and printed the results into text tiles on the Web server. One file listed the IP addresses as a proposed blacklist for an IDS. A second file included data

from all three fields (see Appendix B). The information collected by the PHP sandtrap provided a quick reference point for identifying individual robot sessions in the post-mortem log analysis, and the file in which the script was placed, *clfd.php* provided information about the resource type of PHP for our analysis of the bot traffic.

D. ROBOT EXCLUSION PROTOCOL

Many websites have terms-of-service agreements that explicitly forbid unauthorized scraping or crawling of the website without the permission of its owner. Unfortunately, such terms of service are often ignored by Web robots. Also, the Robots Exclusion Protocol can request that a Web robot not crawl certain areas of a site by specifying restricted areas in a *robots.txt* file [24]. Unfortunately, not all bots cooperate with this voluntary standard, and malicious bots may prefer to target the restricted areas of websites [23]. However, generally Web crawlers used for legitimate purposes check a site's robots.txt before crawling it and abide by the site's exclusion policy [4].

For this reason, we employed a "one-strike" rule to classify bots which accessed the honeypots. If a bot failed to check our site's *robots.txt* file or failed to comply with its directives, it was classified as a "bad" bot. If a bot accessed the honeypot, checked the *robox.txt* file, and followed the directives, it was classified as a "good" bot. A test directive used in the robots.txt file restricted robots from crawling anything the "class" folder:

User-agent:*

Disallow: themes/…/…/…/class/

Figure 1 represents the logic of our classification process in our analysis of bots that accessed our honeypot.

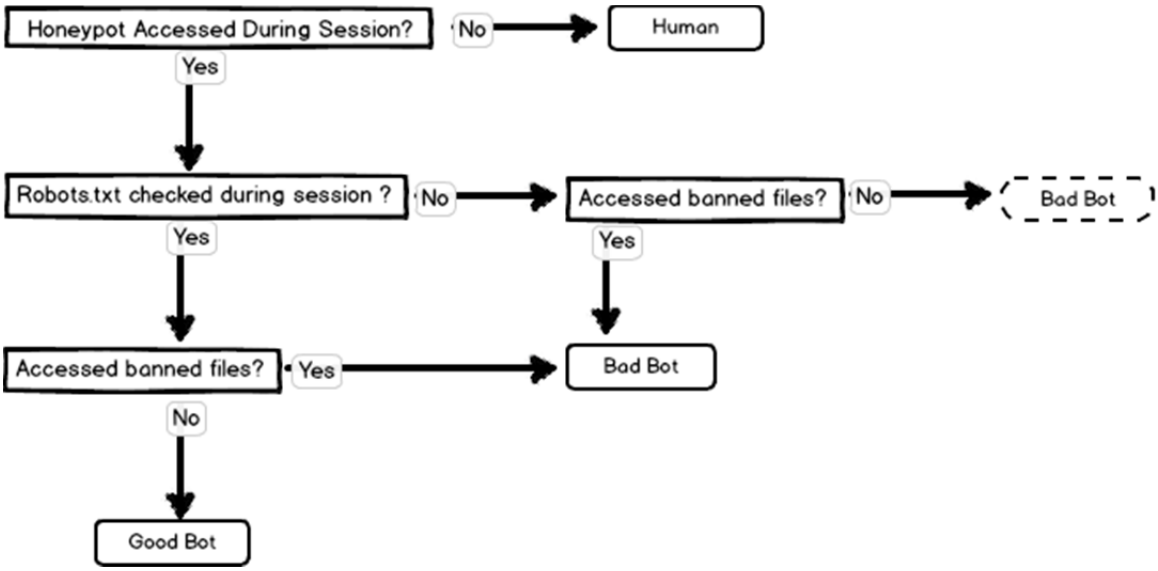


Figure 1. Process of Bot Classification

E. FOCUSED CRAWLING

Focused crawling is a type of crawling that looks at Web page context and link structure of a crawled link before assigning it a download priority. The download priority is based on the likelihood the Web page will lead to the topic being queried by the crawler, so the links with higher download priority are downloaded first. The basic workflow [23] of this process is diagrammed in Figure 2. There a variety of methods used for assigning priority. One common method is to use the similarity of the query to the anchor text of the page link [25]. Anchor text is the text in a hyperlink that is viewable and clickable. For our test, we wanted our honeypot to attract Web bots looking for email so we fashioned the anchor text in the links to include the kinds of email addresses listed in the documents and pages (John.doe@navy.mil, jane.doe@gmail.com). Our assumption was that if crawlers focused on harvesting email addresses visited the NPS library page, they would assign our honeypot links a higher priority.

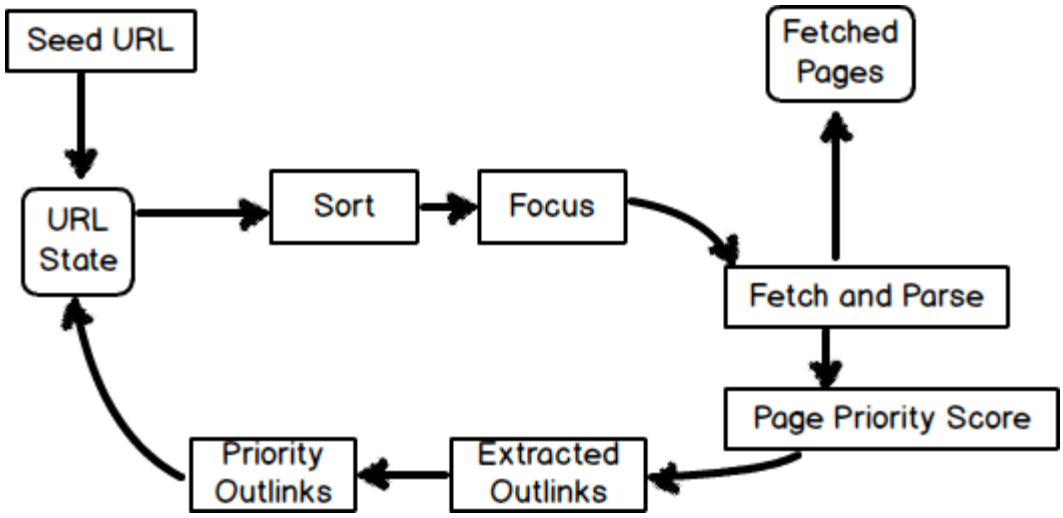


Figure 2. Focused Crawler Workflow

F. BASELINE TEST

We ran a preliminary test using six popular Web crawling and scraping programs. To create the honeypot, we replicated the NPS library homepage "libsearch" on an external Web server. To make a Web architecture and template structure consistent with the library's website, we installed the VuFind open source library portal program on the test server. VuFind is the portal NPS uses for the libsearch.nps.edu website. We then copied the site's homepage template so the test page would resemble the real site (see Appendix C).

The links to the honeypot resource were placed into two different hidden *div* layers within the header template on the test site's index.php. One *div* layer contained links to the honeypot resources in the restricted area (the "class" folder). The other *div* layer contained the links to honeypot resources in a non-restricted area (a "noclass" folder). Appendix D show this inline HTML code which was placed into the header template of the website.

Six crawling programs were selected based upon their popularity and rankings in industry-related blogs. Four of the programs (Import.io, iRobotSoft, 80Legs, and ScrapeBox) were commercial point-and-click tools. The other two were Web scraping frameworks written in the Python and Ruby programming languages.

Import.io is a free Web data extraction service that allows users to extract Web content as they browse Web pages and click on elements in the page. The Import.io program has user-friendly interface that is not conducive to large-scale data extraction since each page must be manually visited before being crawled. It comes with an automatic link-extraction tool. In our test, the link extractor script revealed our "hidden" honey *html* and *php* links, but not the *doc* or *pdf* files. We were able to visit the only the hidden *html* links and harvest the emails on the html files manually. In a review of the Web server access logs, the Import.io service did not identify itself in the user-agent field. Nor did the Import.io crawling program check *robot.txt* nor prevent users from extracting data from restricted areas. However, the robot-exclusion protocol may be correctly ignored due to the fact the user has to visit to the restricted area before a crawler scrapes its data.

80legs is a commercial Web-crawling service provided by the data-mining company Datafiniti. The service provides an online administration portal to users with a list of standard scraping scripts including an email harvester. We ran 80leg's email scraping script our test site. It returned only the email addresses from the *.html* files from the unrestricted area. The 8legs crawler checked the robots.txt file and identified itself in the user-agent field.

Scraperbox is a commercial Web crawling program that identifies itself as "The Swiss Army Knife of SEO." It is a GUI tool designed for increasing Web page search engine optimization (SEO) with a built-in email-scraping tool. It was able to harvest all of the hidden honeypot email addresses except the *pdf* files. It does not check the *robots.txt* file and does not identify itself in the user-agent field.

Scrapy is an open-source Web scraping framework written in Python. We created a Web scraper using it and the BeautifulSoup screen scraping library for Python. We ran two tests search for each class of email address using the following regular expressions:

1) ([a-z0-9][-a-z0-9_\+\.]*[a-z0-9])@([a-z0-9][-a-z0-9\.]*[a-z0-9]\.(mil|gov)|([0-9]{1,3}\.{3}[0-9]{1,3}))

2) ([a-z0-9][-a-z0-9_\+\.]*[a-z0-9])@([a-z0-9][-a-z0-9\.]*[a-z0-9]\.(edu|com)|([0-9]{1,3}\.{3}[0-9]{1,3}))

We employed the PDFMiner3K for parsing the PDF files the scraper was able to collect. Our program did not check the robots.txt file and used a spurious user-agent.

Selenium is a tool that automates Web browsers and was originally developed for website testing [13]. It is a powerful Web scraping tool which it can exercise all of the browser functionality. We used the Selenium `is displayed()` function with our Python crawler and the Firefox browser to exclude all of the fields from our search which were hidden from view. The result returned no emails from our hidden honeypot resources.

Anenome is an open-source Web crawling framework written in the Ruby programming language. Modifying an existing Anenome crawling templates, or "gem," we were able to harvest emails using the same regular expressions from our Python scraper. The anemone crawler was able to collect every email but the ones in the *pdf* files.

G. SUMMARY

Our baseline test demonstrated that all but one of our Web crawlers fetched some of our honeypot resources in their crawls (see Table 3). The Selenium crawler was specifically designed to avoid page elements that were hidden from view and it was able to avoid our honeypot in its crawl. Since Selenium has to execute with an active browser running, it was unclear to whether this crawler could be run efficiently in a large-scale data-mining campaign. Additionally, our results showed the commercial and open-source crawlers we tested which were not commercial services did not check *robots.txt* and limit crawls from restricted areas. All but one of the crawlers were unable to parse *pdf* file types and half of the crawlers we tested could not parse *doc* file types.

Analysis of the access logs indicated that the anchor texts did not encourage our crawlers to download one type of file more frequently than another. We customized crawls to searches for email addresses with either .mil or .com. extensions, but found there were no differences in the resources fetched. However, these results were not unexpected because we did not see any evidence that any that the tool we tested employed focused search method in their crawling algorithms.

Table 3. Summary of Baseline Test Results

Program		Robots.txt		Allowed Resource (class)		Banned Resource (noclass)	
	Checked	pdf	doc	.html	pdf	doc	.html
Import.io	No	No	Yes	Yes	No	Yes	Yes
80Legs	Yes	No	No	Yes	No	No	No
Scrapy	No	Yes	Yes	Yes	Yes	Yes	Yes
Selenium	No	No	No	No	No	No	No
ScrapeBox	No	No	Yes	Yes	No	Yes	Yes
iRobotSoft	No	No	No	Yes	No	No	Yes
Anenome	No	No	Yes	Yes	No	Yes	Yes

H. LIBSEARCH TEST

We embedded the two hidden *div* layers in the header template of every webpage on the NPS library's *libsearch.nps.edu* website. This allowed for the honeypot to be accessible from any point within the website. The honeypots were placed within the VuFind application directory that included the interface for the website. The NPS Library's site did not have a *robots.txt* file, so we added ours to the website's root director. The file included only one directive, which was to restrict crawling of all files within the class folder.

Deep Crawling

The “deep Web” refers to parts of the Web that are not indexed by search engines. The deep Web includes a variety of Web content that is inaccessible to crawling. For example, many shopping and banking website provide structured content which is only accessible through a search interfaces or by logging into the website. These types of Web content are typically more difficult for Web robots to crawl and index. The *libsearch.nps.edu* website has some characteristics of a deep-Web site, and is structured to display majority of its content through its search interface. For a Web robot to access this content, it must be able to generate queries to the website’s search form in its requests. The honeypot resources we placed within the website could not detect a Web robot’s querying of the site’s search forms. However, by placing the honeypot resources within the header portion of the site’s template, we ensured it was accessible from the resulting page of a search query.

V. RESULTS

A. DATA DESCRIPTION

Web logs from the NPS *libsearch.nps.edu* Web server from a five-week period provided the data for our analysis. Our sandtrap logs provided supporting data as well. The data considered for our log analysis were HTTP transactions. HTTP transactions are the requests made by clients, and the corresponding responses from the Web server. These requests are recorded to the Web server's logs, which, for our experiment were Apache's access logs.

During the test period, there were 930,701 HTTP requests, and with an average of 27,373 requests per day. We processed the Web logs to extract the Web robot traffic using the Splunk [24] data analysis program. We applied a syntactic analysis of the "User-Agent" field of the HTTP headers with Splunk's built-in keyword list for denoting bots. Since a user agent string can be easily forged, this process only could identify the self-identifying bots. It does not detect stealth bots that use forged user-agent fields of Web browsers. Table 4 summarizes the aggregate statistics of the percentage of HTTP request of humans and Web robots. Robot traffic on represented 64% of all of the requests on the Libsearch Web server in our study and 18% of the bandwidth consumed.

There were 46 different self-identifying bots which visited the website during this period from a total of 505 different IP addresses. The three major search engines, Bing, Yahoo and Google accounted for of 99% of the search requests (see Appendix E). Approximately 67% of the bot traffic requests consisted of search queries to the */vufind /Search/Results?* page.

Table 4. Summary of Libsearch Web Logs

	Human Traffic	**Robot Traffic**
Total Requests	334,673	596,028
Average Req/Day	9843	17530
Bandwidth Consumed (GB*)*	179.74269	39.4557
% of Distinct Requests	35.955 (36%)	64.040 (64%)

1. Honeypot Results

During our testing period, there were 358 requests for honeypot files on the *nps.libsearch.edu* Web server. Of the requested files, 216 of them were for contents within the unrestricted *noclass* folder, and 142 requests were for content within the restricted *class* folder. The resource request distribution of the *class* and the *noclass* files are represented in Figures 3 and 4.

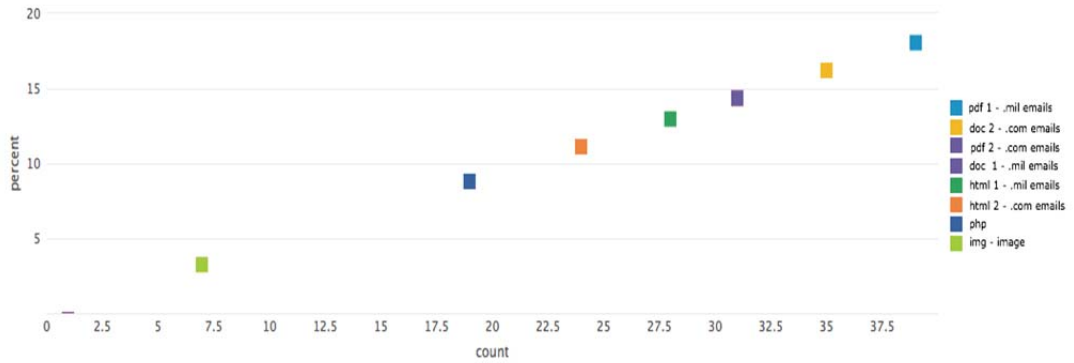


Figure 3. Distribution of Honeypot Requests: Unrestricted Files.

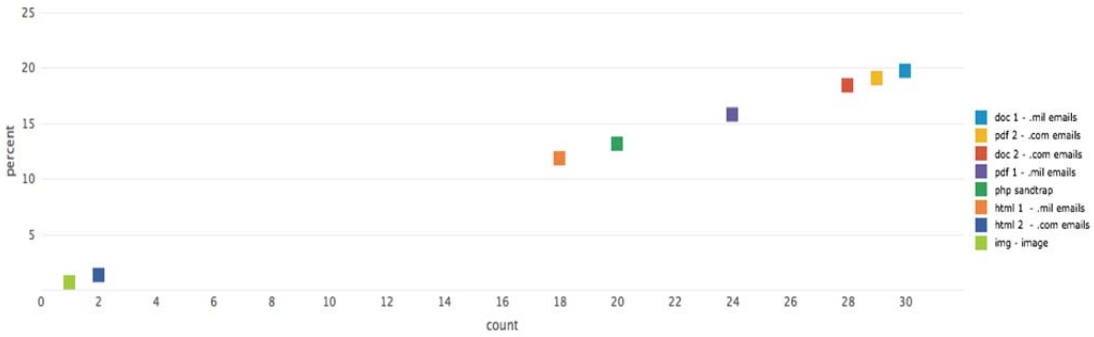


Figure 4. Distribution of Honeypot Requests: Restricted Files

Web robots in our experiment had a higher preference for document resource types (.doc, .html) files than Web resource type (php, html). This contrasted with the findings of Doran [4] which found that Web robots as had a preference for Web resources types over document resource types. There was no significant difference in the number of requests for resources containing civilian email addresses versus resources containing military emails addresses.

In the unrestricted *noclass* folder we observed 21 Web robot campaigns from 59 IP addresses accounting for the 216 HTTP resources requests. A DNS lookup of these IP addresses revealed that 11 of these bots, or 52%, used forged user agent strings (see Appendix E). Ten of the 11 forged user agent string represented Web Browsers and one represented a Google Web robot. Of the 10 self-identified bots, all of them checked the

robots.txt file. Of the 11 bots that used forged user agents, only two had checked the *robots.txt* file.

For the restricted *class* folder, we observed 16 Web robot campaigns from 25 unique IP addresses accounting for 142 HTTP resource request. All 25 of the IP addresses were also in the unrestricted IP list. Seven of the campaigns were self-identified as bots, and a DNS lookup reveal 6 to be accurate with one forged Google bot. The remaining 7 IP addresses used forged user agent fields representing various Web browsers. Only seven of the 16 bots that accessed resources the *class* folder checked the *robots.txt* file. Of the 12 self-identifying bots, only of them, Yandex, came from a well-known search engine. During the experiment, Yandex bots made 548 requests from 13 different IP addresses to the Web server. In total, Yandex accounted for 0.09%, of the total bot traffic on the website. It is unclear why the Russian search engine did not follow the site exclusion protocol. By our classification scheme in Chapter IV, all 25 of these bots of classify as "bad" by not following the site's exclusion protocol.

2. Project Honeypot

We used the http:BL service from the Project Honeypot organization to verify our results from our honeypot test. The Project Honeypot maintains a list of IP addresses of malicious bots that are known to harvest email addresses for spam and other purposes [25]. We checked our list of IP addresses which visited the site during the period of the experiment against the Project Honeypot's list of known malicious bots.

The results of the http:BL lookup produced 40 IPs from the Project Honeypot's blacklist, which accounted for a total of 444 requests on the library's Web server. We compared to our list of 84 IP addresses of bots that requested honeypot resources. We found no matches between our bot list and the Project Honeypot list.

B. ANALYSIS

The results from the honeypot experiment did not produce a confirmation for our hypothesis. The hidden resources in the Web interface did not turn out to be particularly effective in attracting and detecting Web robots on the library's website. Our training test

had provided evidence that honeypots could be valuable for detecting commercial and open source Web crawling programs designed for harvesting emails. In addition, our use of context-specific anchor text in the links seemed to have potential for attracting bots designed to harvest emails with certain domain extensions. However, due to the small sample size in our experiment, we were unable to determine if there was correlation between the honeypot content and the number of requests.

Furthermore, the architecture and content of the website may not have been well suited for honeypot that we used. As we learned through our log analysis, the majority (67%) of bot traffic on the *libsearch.nps.edu* website consisted of queries into the site's catalogues. In hindsight, if would have been beneficial if we had performed an exploratory analysis of the target before determining the location and content of the honeypots. Placing honeypots of the form of spurious query forms would have likely increased the attractiveness of our honeypots.

C. CONCLUSION AND FUTURE RESEARCH

This investigation presented in this thesis focused on the detection of Web robots with honeypots. There is a strong impetus for understanding how honeypots can help detect Web robots, as current methods for real-time detection of Web robots are limited at best. Looking at the data collected from our experiment, it was evident that an exploratory analysis of the Web robot traffic of our test website would have helped to better inform the design of our honeypot. A significant portion of the Web robot requests were queries for deep Web content through the website's search interface. A honeypot using a hidden search form to attract bots might have improved our results. By contrast, our training test showed that honeypots could be an effective tool for detecting crawling, provided the honeypot resource matches the focus of the crawl.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A. PHP CODE FOR THE SANDTRAP

```
<?php
define( “BBOT_ENABLE_LOGGING,” true );
define( “BBOT_LOG_FILE,” dirname( __FILE__ ) . “/bot_log.txt” );
define( “BBOT_BOTLIST,” dirname( __FILE__ ) . “/bot_list.txt” );

function check_ip() {
$botlist = file(BBOT_BOTLIST, FILE_IGNORE_NEW_LINES|
FILE_SKIP_EMPTY_LINES ) );
if ( in_array( $_SERVER[‘REMOTE_ADDR’], $botlist ) ) {
log_ip();
halt();
}
}

function get_ip() { // Put the IP address into the bot_list file
$bfh = fopen( BBOT_BOTLIST, “at” );
fwrite( $bfh, $_SERVER[‘REMOTE_ADDR’] . “\n” );
fclose( $bfh );
log_ip();
                halt();
}

function log_ip() {
if ( BBOT_ENABLE_LOGGING ) { // Put the User Agent, IP address, into bot_log file
$logfh = fopen( BBOT_LOG_FILE, “at” );
fwrite( $logfh,
date(“Y-m-d H:i:s T”) . “;” .
$_SERVER[‘REMOTE_ADDR’] . “;” .
$_SERVER[‘HTTP_USER_AGENT’] .”\n”
);
fclose( $logfh );
}
}

function halt() {
exit(1);
}

check_ip();
?>
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B. EXAMPLE SANDTRAP OUTPUT

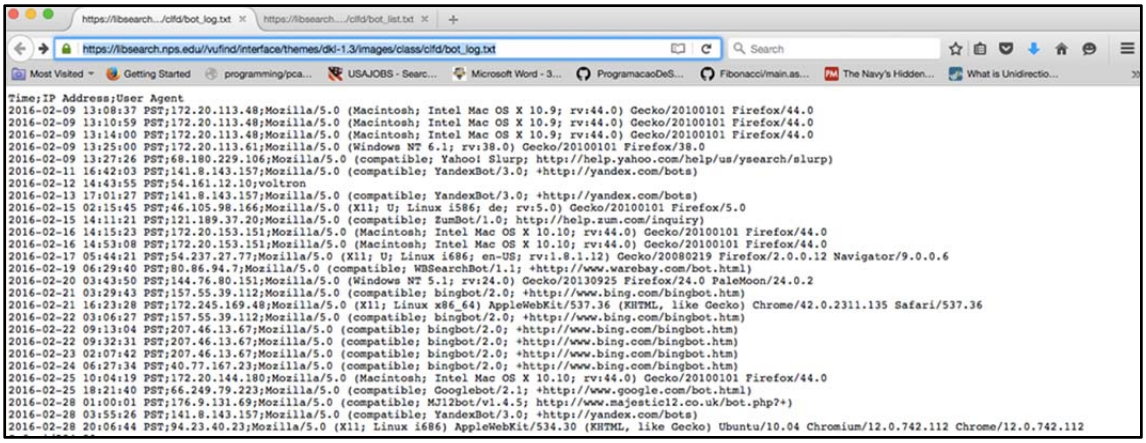


Figure 5. Complete Bot Log

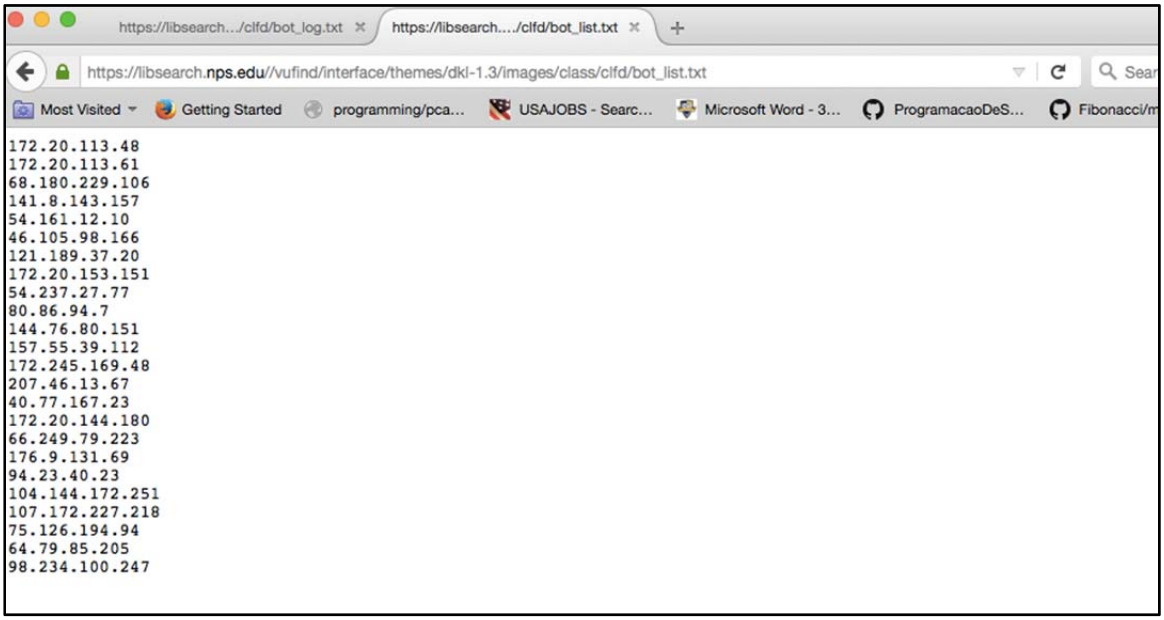
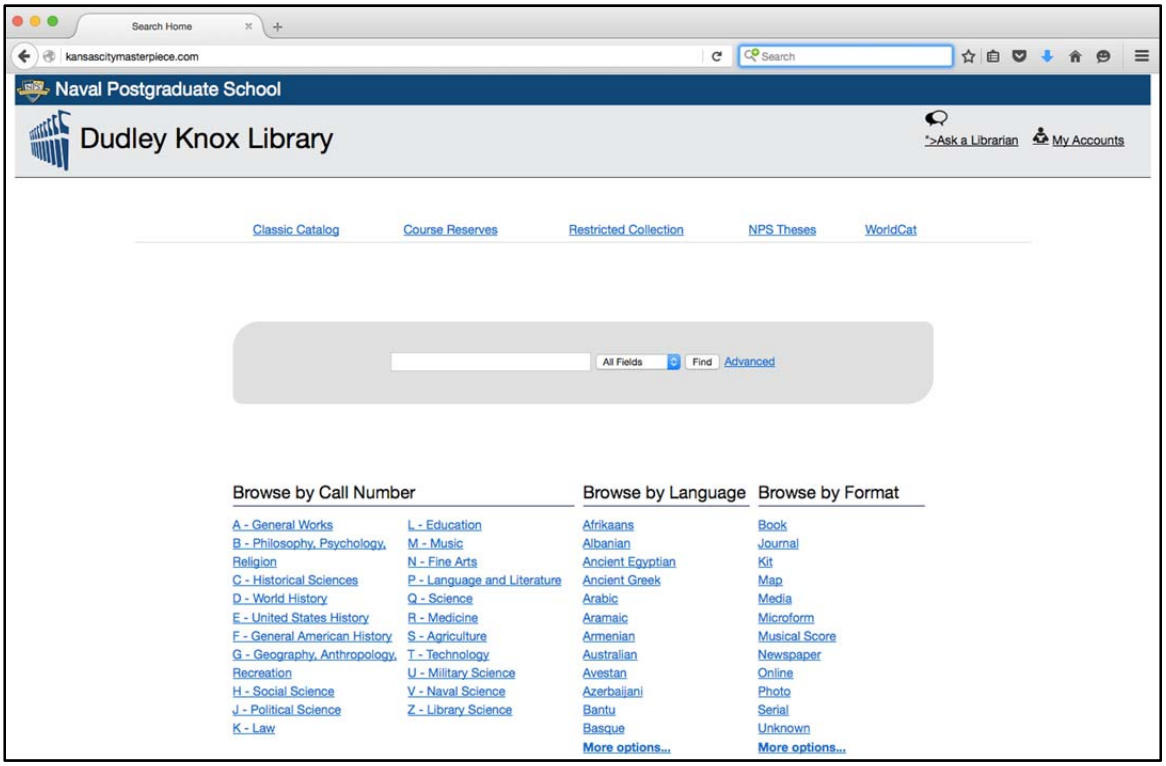


Figure 6. Backlist-ip Bot Log

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX C. TEST SITE APPEARANCE



THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX D. HONEYPOT HEADER TEMPLATE

```
<div id="clas_hp">
<a href="https://libsearch.nps.edu/vufind/interface/themes/dkl-1.3/images/class/clfd.php" style="color: #ffffff">IP</a><br>
<a href="https://libsearch.nps.edu/vufind/interface/themes/dkl-1.3/images/class/CMA/112508.doc" style="color: #ffffff !important; ">John.C.Doe@yahoo.com</a>
<a href="https://libsearch.nps.edu/vufind/interface/themes/dkl-1.3/images/class/CMA/112508.pdf" style="color: #ffffff !important; ">d.cheney@navy.mil</a>
<a href="https://libsearch.nps.edu/vufind/interface/themes/dkl-1.3/images/class/CMA/04151795.pdf" style="color: #ffffff !important; ">hrr22@clintonemail.com</a>
<a href="https://libsearch.nps.edu/vufind/interface/themes/dkl-1.3/images/class/CMA/04151795.doc" style="color: #ffffff !important; ;">s.mck@ncis.navl.mil</a>
<a href="mailto:r.dodge@navy.mil" style="color: #ffffff ">LTC R Dodge</a><br>
<a href="https://libsearch.nps.edu/vufind/interface/themes/dkl-1.3/images/class/CMA/a_mail_clas.html" style="color: #ffffff !important;">a_mail_clas.mil</a>
<a href="https://libsearch.nps.edu/vufind/interface/themes/dkl-1.3/images/class/CMA/a_mail_noclas.html" style="color: #ffffff !important;">a_mail_noclas.com</a>
<img src="Search%20Home_files/dot_clear_a.gif">
<!-- <img src="/vufund/interface/themes/dkl-1.3/images/class/CMA/doc_clear_c.gif"> -->
</div>
<div id="noclas_hp">
<a href="https://libsearch.nps.edu/vufind/interface/themes/dkl-1.3/images/noclass/CMA/112507.doc" style="color: #ffffff !important; ;">s.mck@ncis.navl.mil</a>
<a href="https://libsearch.nps.edu/vufind/interface/themes/dkl-1.3/images/noclass/CMA/112507.pdf" style="color: #ffffff !important; display: none">barck.obama@yahoo.com</a>
<a href="https://libsearch.nps.edu/vufind/interface/themes/dkl-1.3/images/noclass/CMA/04151796.pdf" style="color: #ffffff !important; ">sken@cia.gov</a>
<a href="https://libsearch.nps.edu/vufind/interface/themes/dkl-1.3/images/noclass/CMA/04151796.doc" style="color: #ffffff !important; ">bob.g.allen@gmail.com</a>
<a href="https://libsearch.nps.edu/%3C-%21mailto:tester@navy.mil" style="color: #ffffff">cirodesk@nps.edu</a><br>
<a href="https://libsearch.nps.edu/vufind/interface/themes/dkl-1.3/images/noclass/CMA/b_mail_clas.html" style="color: #ffffff !important;">b_mail_clas.mil</a>
<a href="https://libsearch.nps.edu/vufind/interface/themes/dkl-1.3/images/noclass/CMA/b_mail_noclas.html" style="color: #ffffff !important;">b_mail_noclas.com</a>
<a href="https://libsearch.nps.edu/vufind/interface/themes/dkl-1.3/images/noclass/emails.txt" style="color: #ffffff">email list</a>
<!--<img src="/vufind/interface/themes/dkl-1.3/images/noclass/CMA/dot_clear_b.gif">-->
</div>
</div>
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX E. BOT CAMPAIGNS

Table 5. Self-Identified Crawler Campaigns—Total

Web Robot	Requests	Percent	IPs
Bingbot	494911	70.419892	197
Yahoo! Slurp;	111459	15.859277	10
Googlebot/2.1;	92658	13.184121	29
Baiduspider/2.0;	1445	0.205606	121
Googlebot/2.1; -iPhone	659	0.093768	8
YandexBot/3.0;	548	0.077974	13
MJ12bot	487	0.069294	19
DuckDuckGo	204	0.029027	1
Applebot/0.3	55	0.007826	2
bingbot/2.0 - iPhone	38	0.005407	23
Googlebot-Image/1.0	36	0.005122	4
YandexImages/3.0	32	0.004553	9
Google Web Preview Analytics	31	0.004411	1
Exabot/3.0	20	0.002846	1
Googlebot (gocrawl v0.4)	19	0.002703	19
WBSearchBot	16	0.002277	1
ZumBot/1.0; http://help.zum.com/inquiry)	14	0.001992	1
Applebot/0.1;	12	0.001707	1
Findxbot/1.0;	12	0.001707	1
SputnikBot	10	0.001423	5
Cliqzbot/1.0	10	0.001423	4
bingbot/2.0;	8	0.001138	3
Mail.RU_Bot	7	0.000996	5
yacybot	6	0.000854	2
Speedy Spider	6	0.000854	1
MS Search 6.0 Robot	5	0.000711	1
SeznamBot/3.2;	4	0.000569	2
SemrushBot/1~bl;	4	0.000569	2
AhrefsBot/5.0;	4	0.000569	2
Domain Re-Animator Bot	4	0.000569	1
CSS Certificate Spider	4	0.000569	1
bhcBot	4	0.000569	1
archiver/3.1.1	3	0.000427	1
Applebot/0.1 -iPhone	2	0.000285	1
linkdexbot/2.2;	2	0.000285	1
BLEXBot/1.0;	2	0.000285	1

Web Robot	Requests	Percent	IPs
archive.org_bot	2	0.000285	1
Googlebot/2.1	2	0.000285	2
Crawler@alexa.com	2	0.000285	1
ZumBot/1.0;	1	0.000142	1
SurdotlyBot/1.0;	1	0.000142	1
SputnikImageBot/2.3	1	0.000142	1
PrivacyAwareBot/1.1;	1	0.000142	1
ParsijooBot;	1	0.000142	1
linkapediabot	1	0.000142	1

Table 6. Bots Accessing Unrestricted "noclass" Folder

DNS Lookup	Bot/User-Agent	IP	Requests
msnbot-157-55-39-162.search.msn.com*	bingbot/2.0;	26	38
crawl-66-249-79-223.googlebot.com	Googlebot/2.1	1	14
static.69.131.9.176.clients.your-server.de	MJ12bot/v1.4.5;	2	6
5e.c2.7e4b.ip4.static.sl-reverse.com	ScoutJet;	1	1
loft1165.serverloft.com	WBSearchBot	1	2
b115340.yse.yahoo.net	Yahoo! Slurp;	1	16
spider-141-8-143-157.yandex.com*	YandexBot/3.0;	3	86
no results	ZumBot/1.0;	1	3
crawl-66-249-79-223.googlebot.com	Googlebot/2.1 - iPhone	3	9
no results	Mozilla/5.0 (Macintosh; Intel Mac OS X 10.10; rv:38.0)	2	11
ec2-54-237-27-77.compute-1.amazonaws.com	Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10.6; en-US; rv:1.9.2.13;) Gecko/20101203	1	1
static.151.80.76.144.clients.your-server.de	Mozilla/5.0 (Windows NT 5.1; rv:24.0) Firefox/24.0 PaleMoon/24.0.2	1	2
no results	Mozilla/5.0 (Windows NT 6.3; WOW64; rv:38.0) Gecko/20100101 Firefox/38.0	1	1
ec2-54-237-27-77.compute-1.amazonaws.com	Mozilla/5.0 (Windows; U; Windows NT 6.1; rv:2.2) Gecko/20110201	1	2
crawl15.lp.007ac9.net	Mozilla/5.0 (X11; Linux i686) AppleWebKit/534.30 (KHTML, like Gecko)…	1	2
107-172-229-131-host.colocrossing.com	Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36	1	5
accomodationhub.com	Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36	2	3
technology-approval.com	Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36	1	2
crawl20.lp.007ac9.net	Mozilla/5.0 (X11; U; Linux i586; de; rv:5.0) Gecko/20100101 Firefox/5.0	1	2
172-245-169-14-host.colocrossing.com	Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:38.0) Gecko/20100101 Firefox/38.0	1	3
ec2-54-80-244-153.compute-1.amazonaws.com	voltron	7	7
	Totals	59	216

Table 7. Bots Accessing Restricted Folder

DNS Lookup	Agent	IP	Requests
spider-100-43-85-10.yandex.com	**Mozilla/5.0 YandexBot/3.0;**	2	2
technology-approval.com	**Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/47.0.2526.111 Safari/537.36**	1	2
accomodationhub.com	**Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/47.0.2526.111 Safari/537.36**	2	3
107-172-227-218-host.colocrossing.com	**Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/47.0.2526.111 Safari/537.36**	3	5
spider-141-8-143-157.yandex.com	**YandexBot/3.0**	2	62
static.151.80.76.144.clients.your-server.de	**Mozilla/5.0 (Windows NT 5.1; rv:24.0) Gecko/20130925 Firefox/24.0 PaleMoon/24.0.2**	1	3
172-245-169-48-host.colocrossing.com	**Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/42.0.2311.135 Safari/537.36**	1	1
static.69.131.9.176.clients.your-server.de	**Mozilla/5.0 (compatible; MJ12bot/v1.4.5; http://www.majestic12.co.uk/bot.php?+)**	1	6
crawl20.lp.007ac9.net	**Mozilla/5.0 (X11; U; Linux i586; de; rv:5.0) Gecko/20100101 Firefox/5.0**	1	19
ec2-54-158-20-81.compute-1.amazonaws.com	**voltron**	1	3
ec2-54-237-27-77.compute-1.amazonaws.com	**Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.8.1.12) Gecko/20080219 Firefox/2.0.0.12 Navigator/9.0.0.6**	1	3
no result	**Mozilla/5.0 (Macintosh; Intel Mac OS X 10.9; rv:44.0) Gecko/20100101 Firefox/44.0**	1	2
no results	**Mozilla/5.0 (Windows NT 6.1; rv:38.0) Gecko/20100101 Firefox/38.0illa/5.0 (compatible; Googlebot/2.1; +http://www.google.com/bot.html)**	3	20
5e.c2.7e4b.ip4.static.sl-reverse.com	**ScoutJet;**	1	1
loft1165.serverloft.com	**WBSearchBot/1.1; +http://www.wareBay.com/bot.html)**	1	5
crawl15.lp.007ac9.net	**Mozilla/5.0 (X11; Linux i686) AppleWebKit/534.30 (KHTML, like Gecko) Ubuntu/10.04 Chromium/12.0.742.112 Chrome/12.0.742.112 Safari/534.30**	1	3
	Totals	**25**	**142**

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

[1] F. Li, K. Goseva-Popstojanova, and A. Ross, "Discovering web workload characteristics through cluster analysis," in *Proc. IEEE International Symposium on Network Computing and Applications*, 2007, Cambridge, MA.

[2] *Distil Networks annual bad bot landscape report*. (2012, Mar. 14) [Online]. Available: http://resources.distilnetworks.com/h/i/53361151-2014-bad-bot-landscape-report/185088

[3] D. Shestakov, "Intelligent Web Crawlers," *IEEE Intelligent Informatics Bulletin*. vol.14, no.1, Dec. 2013.

[4] D. Doran, K Morillo, and. S.S Gokhale, "A comparison of Web robot and human requests," in *IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM),* Niagra, Ontario , CAN, 2013, pp. 25–28.

[5] M. Calzarossa, L. Massari and D. Tessera, "An extensive study of Web robots traffic." In *Proceedings of International Conference on Information Integration and Web-based Applications & Services (IIWAS '13).* New York, NY, 2013.

[6] D. Doran and S. S. Gokhale. "Web robot detection techniques: Overview and limitations." *Data Mining and Knowledge Discovery,* vol. 22, no. 1, pp. 183–210, 2009

[7] G. Jain and Y. Mehandiratta, "Chronicle security against covert crawling. " In *Proceedings of the First International Conference on Security of Internet of Things*, 2012, New York, NY.

[8] A. Koehl and H. Wang. "Surviving a search engine overload." In *Proceedings of the 21st international conference on World Wide Web (WWW '12). ACM*, 2012 New York, NY, pp. 171–180.

[9] M. Smith, (2015 May 13*). Akamai report: DDoS attacks doubled in Q1 2015, SSDP top attack vector* [Online]. Available: http://www.networkworld.com/article/2925152/microsoft-subnet/akamai-report-ddos-attacks-doubled-in-q1-2015-ssdp-top-attack-vector.html.

[10] L. Qin, L. Hong, K. Songlin, L. Chuchu, "Feature extraction and construction of application layer DDoS attack based on user behavior." *Proceedings of the 33rd Chinese Control Conference (CCC)*, 2014, pp. 5492–5497.

[11] G. Xie, H. Hang, and M. Faloutsos. “Scanner hunter: Understanding HTTP scanning traffic.” In *Proceedings of the 9th ACM Symposium on Information, Computer and Communications Security (ASIA CCS ‘14)*, 2014, New York, NY, pp. 27–38.

[12] P. Ducklin, (2013 Jan. 7). *DHS website falls victim to hacktivist intrusion* [Online]. Available: http://nakedsecurity.sophos.com/2013/01/07/dhs-website-falls-victim-to-hacktivist-intrusion/

[13] R. Mitchell, *Web Scraping with Python: Collecting Data from the Modern Web*. Sebastopol, CA: O’Reilly Media, 2015.

[14] “eBay, Inc. v. Bidder’s Edge, Inc.” (2016, Feb. 10) [Online]. Available: http://www.tomwbell.com/NetLaw/Ch06/eBay.html.

[15] C. Shu, (2013 Apr. 30). *Judge Throws Out Craigslist’s Copyright Lawsuit, But It Can Still Sue 3Taps Over Data Use* [Online]. Available: http://techcrunch.com/2013/04/30/craigslist-3taps-lawsuit-decision/http://techcrunch.com/2013/04/30/craigslist-3taps-lawsuit-decision/.

[16] A, Brooks. (2015 May 3) *How Selerity Reported Twitter’s Earnings—before Twitter Did* [Online]. Available: http://arstechnica.com/business/2015/05/how-selerity-reported-twitters-2q15-earnings-before-twitter-did/.

[17] D. Yegulalp. (2013 Nov 8) *Google’s dangerous bots put the whole Web on edge* [Online]. Available: http://www.infoworld.com/article/2609489/security/google-s-dangerous-bots-put-the-whole-web-on-edge.html.

[18] *Make sure Googlebot is not blocked - Search Console Help* [Online]. Available: https://support.google.com/webmasters/answer/2387297?hl=en.

[19] A. Stassopoulou and M. Dikaiakos, “Web robot detection: A probabilistic reasoning approach” *Comput. Networks*, vol. 53, no. 3. pp. 265–278. Feb. 2009.

[20] D. Doran, “Detection, Classification, and Workload Analysis of Web Robots” Ph.D. dissertation, Dept. Comp. Sci., Univ. of Conneticut, Storrs, CT, 2014.

[21] K. Park, V. Pai, K. Lee, and S. Calo.”Securing web service by automatic robot detection.” *in Proceedings of the annual conference on USENIX ‘06 Annual Technical Conference (ATEC ‘06)*, Berkeley, CA, 2006.

[22] Chu, S. Gianvecchio, A. Koehl, H. Wang, and S. Jajodia, “Blog or block: Detecting bots through behavioral biometrics,” *Comp. Networks*, vol. 57, pp. 634–646, 2013

[23] C. Olston and M. Najork, “Web crawling,” *Foundations and Trends in Information Retrieval*, vol. 4, no. 3, pp. 175–246, 2010.

[24] W. Di, L. Tian, B. Yan, W. Liyuan, and Li Yanhui, “Study on SEO monitoring system based on keywords & links,” in *Computer Science and Information Technology (ICCSIT), 3rd IEEE International Conference on*, Chengdu, China 2010, pp. 450–453.

[25] S. Batsakis, E.G. Petrakis, E. Milios, “Improving the performance of focused web crawlers,” *Data & Knowledge Engineering*, vol.68, No.10, pp.1001–1013, 2010.

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California